

# The Daala Directional Deringing Filter

Jean-Marc Valin\*

April 8, 2016

## Abstract

This paper presents the deringing filter used in the Daala royalty-free video codec. The filter is based on a non-linear conditional replacement filter and is designed for vectorization efficiency. It takes into account the direction of edges and patterns being filtered. The filter works by identifying the direction of each block and then adaptively filtering along the identified direction. In a second pass, the blocks are also filtered in a different direction, with more conservative thresholds to avoid blurring edges. The proposed deringing filter is shown to improve the quality of both Daala and the Alliance for Open Media (AOM) AV1 video codec.

## 1 Introduction

The main goal of deringing is to filter out ringing, while retaining all the details of the image. The amount of ringing tends to be roughly proportional to the quantization step size. The amount of detail is a property of the input image, but the smallest detail actually retained in the decoded image tends to be roughly proportional to the quantization step size. For a given quantization step size, the amplitude of the ringing is generally less than the amplitude of the details.

A standard linear filter works by averaging nearby values of the input to produce an output. For example, a 3-tap filter can be as simple as averaging the sample  $x(n)$  being filtered with its two neighbors:  $y(n) = (x(n-1) + x(n) + x(n+1))/3$ . This works well when we want to eliminate all high frequencies, but in the video context, it has the side effect of blurring out all details.

This paper describes a deringing filter that takes into account the direction of edges and patterns being filtered. The filter works by identifying the direction of each block and then adaptively filtering along the identified direction. In a second pass, the blocks are also filtered in a different direction, with more conservative thresholds to avoid blurring edges. The deringing filter is also designed to be easy to vectorize, requiring no per-pixel scalar operation. An high-level interactive demonstration of the algorithm is available at [\[1\]](#).

## 2 Direction Search

The first step is to divide the image into blocks of fixed or variable size. Variable-size blocks make it possible to use large blocks on long, continuous edges and small blocks where edges intersect or change direction. A fixed block size is easier to implement and does not require signaling the sizes on a block-by-block basis. For this work, we consider a fixed block size of 8x8 since it is fine enough to follow non-straight edges, but large enough to reliably estimate directions.

Once the image is divided into blocks, we determine which direction best matches the pattern in each block. One way to determine the direction is to minimize sum of squared differences (SSD) between the input block and a perfectly directional block. A perfectly directional block is a block for which each line along a certain direction has a constant value. For each direction, we assign a line number to each pixel, as shown in Fig. 1.

For each direction  $d$ , the SSD is defined as:

$$\sigma_d^2 = \sum_{k \in \text{block}, d} \left[ \sum_{p \in P_{d,k}} (x_p - \mu_{d,k})^2 \right], \quad (1)$$

---

\*Copyright 2014-2016 Mozilla Foundation. This work is licensed under [CC-BY 4.0](#). Send correspondence to Jean-Marc Valin [<jmvalin@jmvalin.ca>](mailto:jmvalin@jmvalin.ca).

0	0	1	1	2	2	3	3
1	1	2	2	3	3	4	4
2	2	3	3	4	4	5	5
3	3	4	4	5	5	6	6
4	4	5	5	6	6	7	7
5	5	6	6	7	7	8	8
6	6	7	7	8	8	9	9
7	7	8	8	9	9	10	10

Figure 1: Line numbers for pixels following one direction in an 8x8 block.

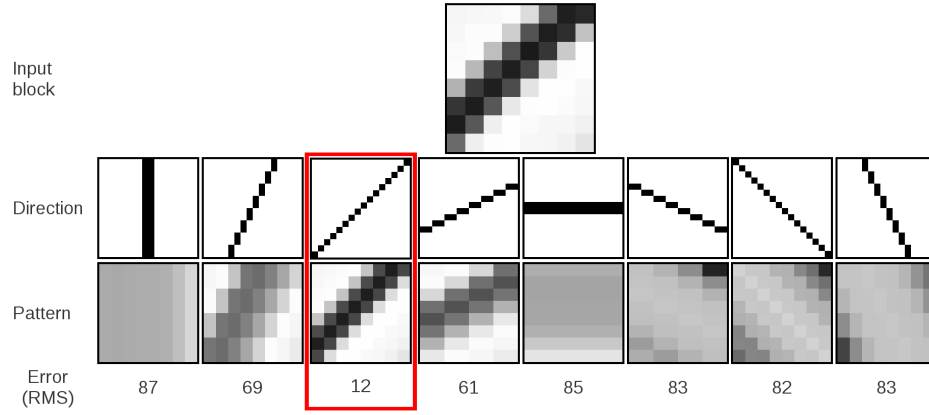


Figure 2: Example of direction search for an 8x8 block. The patterns shown are based on the  $\mu_{d,k}$  values. In this case, the 45-degree direction is the one that minimizes  $\sigma_d^2$ , so it would be selected by the search. Note that the error values  $\sigma_d$  shown are never computed in practice (only  $s_d$  is).

where  $x_p$  is the value of pixel  $p$ ,  $P_{d,k}$  is the set of pixels in line  $k$  following direction  $d$ , and  $\mu_k$  is the pixel average for line  $k$ :

$$\mu_{d,k} = \frac{1}{N_{d,k}} \sum_{p \in P_{d,k}} x_p, \quad (2)$$

where  $N_{d,k}$  is the cardinality of  $P_{d,k}$ . Substituting (2) into (1) and simplifying, we get

$$\sigma_d^2 = \sum_{p \in \text{block}} x_p^2 - \sum_{k \in \text{block}, d} \frac{1}{N_{d,k}} \left( \sum_{p \in P_{d,k}} x_p \right)^2, \quad (3)$$

Considering that the first term of Eq. (3) is constant with respect to  $d$ , we simply find the optimal direction  $d_{opt}$  by maximizing the second term:

$$d_{opt} = \max_d s_d, \quad (4)$$

where

$$s_d = \sum_{k \in \text{block}, d} \frac{1}{N_{d,k}} \left( \sum_{p \in P_{d,k}} x_p \right)^2. \quad (5)$$

Fig. 2 shows an example of a direction search for an 8x8 block containing a line.

Input	26	8	22	25	24	23	80	Average = 29
Replacement mask for $T = 5$	1	0	1	1	1	1	0	
Input after replacement	26	25	22	25	24	23	25	Average = 24

Figure 3: Conditional replacement filter computation

### 3 Conditional Replacement Filter

Just like the median filter [2] and the bilateral filter [3], the conditional replacement filter is designed to remove noise without blurring sharp edges. However, it is simpler to compute and is easier to vectorize than the median filter or the bilateral filter. A regular linear filter with  $(2M + 1)$  taps is defined as

$$y(n) = \frac{1}{W} \sum_{k=-M}^{k=M} w_k x(n+k) , \quad (6)$$

where  $W = \sum_{k=-M}^M w_k$ .

The main difference between a regular filter and the conditional replacement filter is that for each tap, if  $x(n+k)$  differs from  $x(n)$  by more than a threshold  $T$ , then we use  $x(n)$  instead for the tap. The filter computation is illustrated in Fig. 3 and an example is shown in Fig. 4.

Through algebraic simplifications, the filter definition can be written in terms of the differences  $x(n+k) - x(n)$ , which yields

$$y(n) = x(n) + \frac{1}{W} \sum_{k=-M, k \neq 0}^{k=M} w_k f(x(n+k) - x(n), T) , \quad (7)$$

with the threshold function

$$f(d, T) = \begin{cases} d & , |d| < T \\ 0 & , \text{otherwise} \end{cases} . \quad (8)$$

The advantage of this formulation is that the normalization by  $\frac{1}{W}$  can be approximated without causing any bias, even when  $W$  is not a power of two. Also, because  $W$  does not depend on the number of pixels being replaced, the normalization is easy to vectorize over a row (or column) of pixels.

#### 3.1 Directional Filtering

The directional filter for pixel  $(i, j)$  is defined as the 7-tap conditional replacement filter

$$y(i, j) = x(i, j) + \frac{1}{W} \sum_{k=1}^3 w_k [f(x(i, j) - x(i + \lfloor kd_y \rfloor, j + \lfloor kd_x \rfloor), T_d) + f(x(i, j) - x(i - \lfloor kd_y \rfloor, j - \lfloor kd_x \rfloor), T_d)] \quad (9)$$

where  $d_x$  and  $d_y$  define the direction,  $W$  is a constant normalizing factor,  $T_d$  is the filtering threshold for the block. The direction parameters are shown in Table 1. The weights  $w_k$  can be chosen so that  $W$  is a power of two. For example, Daala currently uses  $\mathbf{w} = [3 \ 2 \ 2]$  with  $W = 16$ . Since the direction is constant over 8x8 blocks, all operations in this filter are directly vectorizable over the blocks.

#### 3.2 Second Stage Filter

The 7-tap directional filter is sometimes not enough to eliminate all ringing, so we use an additional filtering step that operates across the direction lines used in the first filter. Considering that the input of the second filter has considerably less ringing than the input of the second filter, and the fact that the second filter risks blurring edges, the position-dependent threshold  $T_2(i, j)$  for the second filter is set lower than that of the first filter  $T_d$ . The filter

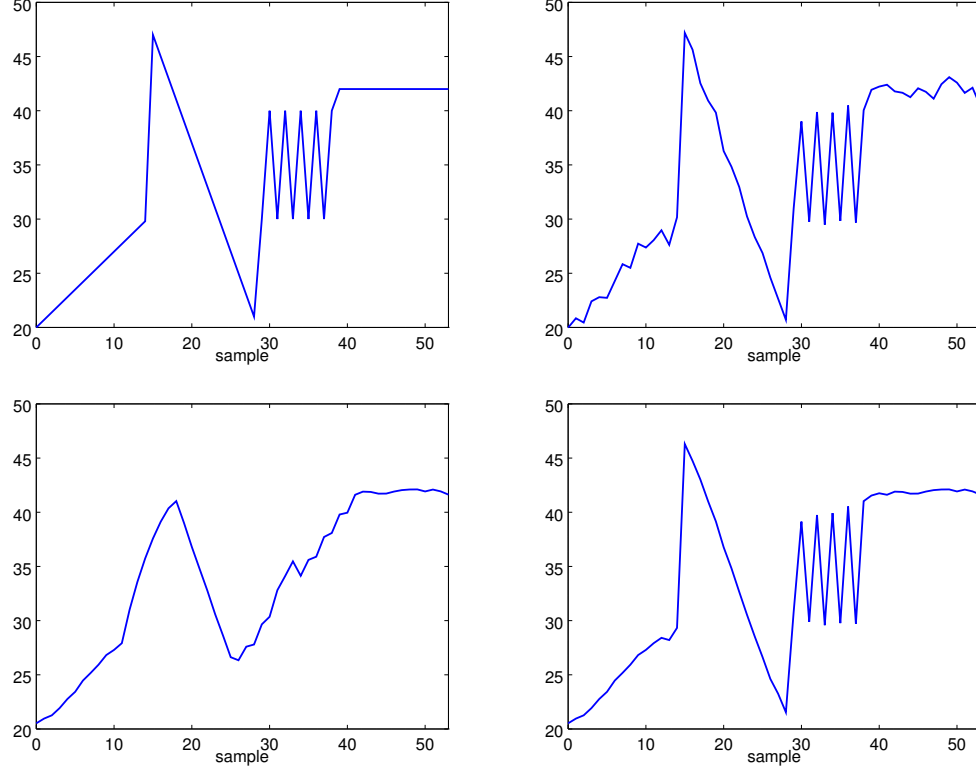


Figure 4: Conditional replacement filter example. Up-left: original signal, up-right: noisy signal, bottom-left: filtered with 7-tap linear filter, bottom-right: filtered with 7-tap conditional replacement filter.

Direction	$d_x$	$d_y$
0	1	-1
1	1	$-1/2$
2	1	0
3	1	$1/2$
4	1	1
5	$1/2$	1
6	0	1
7	$-1/2$	1

Table 1: Direction parameters

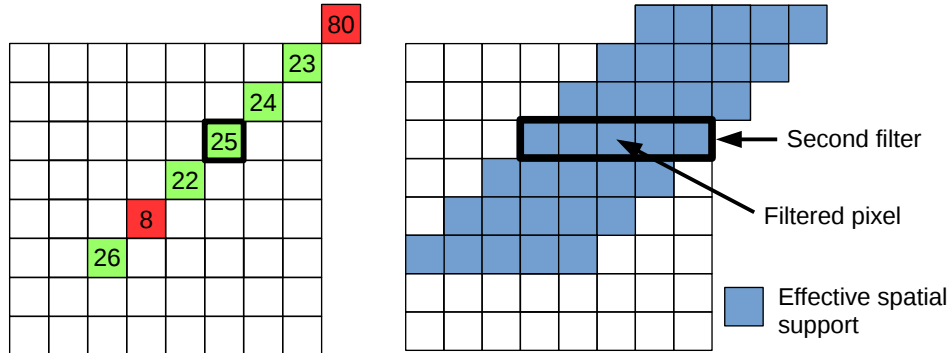


Figure 5: Filtering along the direction (left) and filtering across the direction (right).

Direction	$d_x$	$d_y$
0	1	0
1	0	1
2	0	1
3	0	1
4	1	0
5	1	0
6	1	0
7	1	0

Table 2: Second stage filter parameters

structure is the same as the one in Eq. (9). The direction parameters for the second stage filter are shown in Table (2) and the filter weights are  $\mathbf{w} = \begin{bmatrix} 1 & 1 \end{bmatrix}$  with  $W = 16/3$ . Considering that each input pixel from the second filter is itself the output of the 7-tap directional filter, the combination of the two filters is effectively a 35-tap separable filter.

## 4 Setting Thresholds

The thresholds  $T_d$  and  $T_2$  must be set high enough to smooth out ringing artifacts, but low enough to avoid blurring important details in the image. Although the ringing is *roughly* proportional to the quantization step size  $Q$ , as the quantizer increases the error grows slightly less than linearly because the unquantized coefficients become very small compared to  $Q$ . As a starting point for determining the thresholds, Daala uses a power model of the form

$$T_0 = \alpha_1 Q^\beta \ell, \quad (10)$$

with  $\beta = 0.842$  in Daala, and where  $\alpha_1$  depends on the input scaling. The deringing *level*  $\ell$  is a threshold adjustment coded for each superblock (64x64). In the AV1 codec, a global threshold is selected by the encoder instead of using a function of the quantizer, so

$$T_0 = \ell_g \cdot \ell \quad (11)$$

Another factor that affects the optimal filtering threshold is the presence of strong directional edges/patterns. These can be estimated from the  $s_d$  parameters computed in Eq. (5) as

$$\delta = s_{d_{opt}} - s_{d_{ortho}}, \quad (12)$$

where  $d_{ortho} = d_{opt} + 4 \pmod{8}$ . We compute the direction filtering threshold for each block as

$$T_d = T_0 \cdot \max\left(\frac{1}{2}, \min\left(3, \alpha_2 \delta^{1/6}\right)\right), \quad (13)$$

where  $\alpha_2$  also depends on the input scaling. For the second filter, we use a more conservative threshold that depends on the amount of change caused by the directional filter.

$$T_2(i, j) = \min\left(T_d, \frac{T_d}{3} + |y(i, j) - x(i, j)|\right). \quad (14)$$

As a special case, when the pixels corresponding to the 8x8 block being filtered are all skipped, then  $T_d = T_2 = 0$ , so no deringing is performed.

## 5 Superblocks and Signaling

The filtering is applied one superblock at a time, in a way that depends on the level  $\ell$ . The level can take one of 6 values:

$$\ell \in \{0, 0.5, 0.7, 1.0, 1.4, 2.0\}, \quad (15)$$

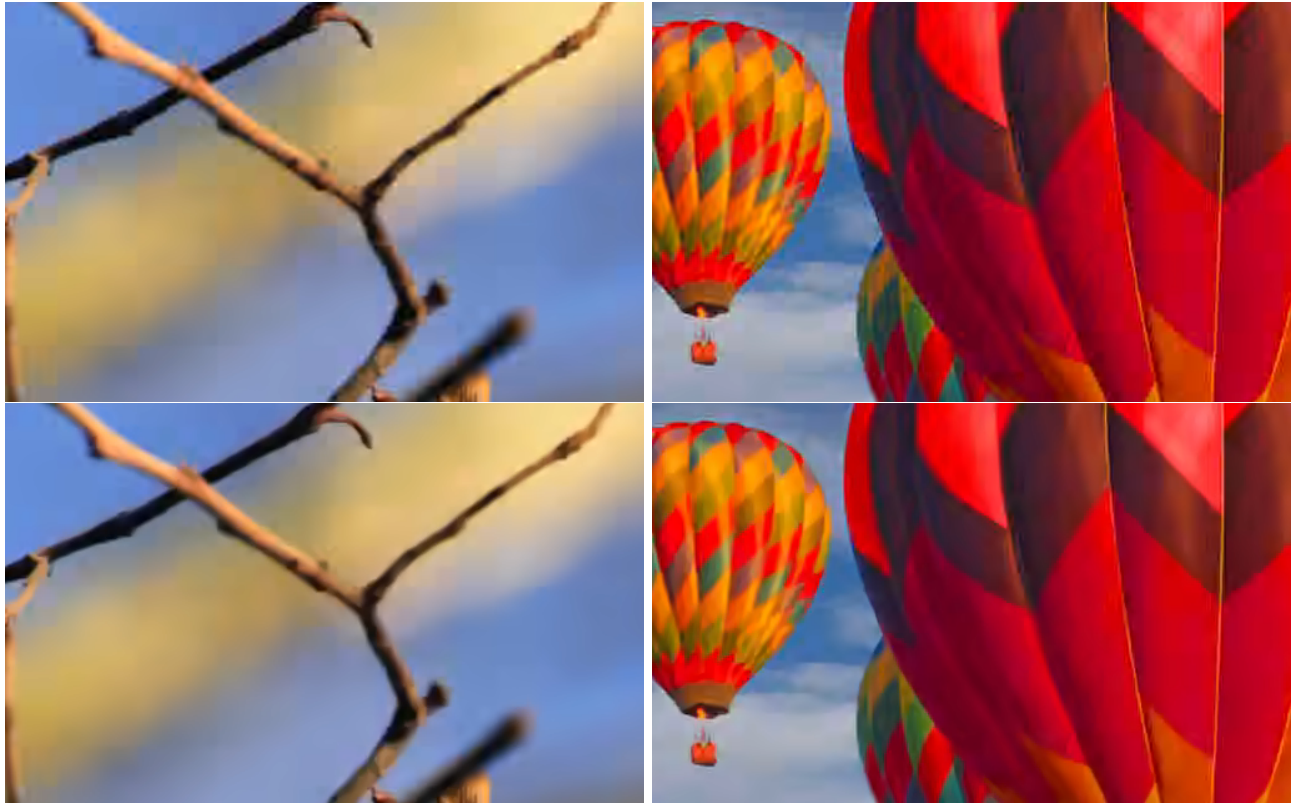


Figure 6: Visual effect of deringing at low bitrate for Daala. Top: without deringing, bottom: with deringing.

where  $\ell = 0$  disables the deringing filter for the current superblock. The level is the only information coded in the bitstream by the deringing filter. On keyframes, it is entropy-coded based on the neighbor values. On inter-predicted frames, the level is only coded for superblocks that are not skipped and is entropy-coded based on a single adapted probability distribution (no context from the neighbors). Superblocks where no level is coded have deringing disabled. Similarly, any skipped block within a superblock has deringing disabled, even if it is signaled enabled for the superblock.

The level of the deringing filter in AV1 is handled similarly, except that only four levels are currently available and there is no entropy coding yet.

The deringing process sometimes reads pixels that lie outside of the superblock being processed. When these pixels belong to another superblock, the filtering always uses the unfiltered pixel values – even for the second stage filter – so that no dependency is added between the superblocks. This makes it possible to filter all superblocks in parallel. When the pixels used for a filter lie outside of the viewable image, we set  $f(d, T) = 0$  in Eq. (8).

## 6 Results

The deringing filter described here has been implemented for the Daala [4] codec and is available in the master branch of the Daala Git repository [5]. Its implementation lies in the `src/dering.c` file. An example of the effect of the deringing filter at low bitrate on a still image is shown in Fig. 6.

We tested the deringing filter using the Are We Compressed Yet? [6] online testing tool. The results for still images are shown in Table 3 for the subset1 test set and those for video are shown in Table 4 for the ntt-short1 test set. Visual inspection confirms that the quality is greatly improved, despite the regression in the FAST-SSIM results.

Bitrate (bpp)	PSNR (%)	PSNR-HVS (%)	SSIM (%)	FAST-SSIM (%)
0.1 – 0.2	-3.5	-2.8	-1.6	+2.6
0.2 – 0.5	-2.9	-2.2	-1.6	+3.2
0.5 – 1	-1.7	-0.9	-1.0	+3.6

Table 3: Deringing filter Bjøntegaard-delta [7] rate for still images (lower is better) in Daala.

Bitrate (bpp)	PSNR (%)	PSNR-HVS (%)	SSIM (%)	FAST-SSIM (%)
0.005 – 0.02	-5.8	-4.3	-4.0	+7.2
0.02 – 0.06	-7.4	-4.7	-5.8	+11.5
0.06 – 0.2	-7.9	-5.0	-7.7	+12.2

Table 4: Deringing filter Bjøntegaard-delta [7] rate for video sequences (lower is better) in Daala.

Bitrate (bpp)	PSNR (%)	PSNR-HVS (%)	SSIM (%)	FAST-SSIM (%)
0.2 – 0.5	-2.0	-1.4	-0.9	+2.8
0.5 – 1	-0.9	-0.5	-0.6	+1.5

Table 5: Deringing filter Bjøntegaard-delta [7] rate for still images (lower is better) in AOM codec.

Bitrate (bpp)	PSNR (%)	PSNR-HVS (%)	SSIM (%)	FAST-SSIM (%)
0.02 – 0.06	-2.5	-1.5	-1.5	+3.8
0.06 – 0.2	-2.0	-0.8	-1.3	+3.1

Table 6: Deringing filter Bjøntegaard-delta [7] rate for video sequences (lower is better) in AOM codec.

## 7 Conclusion

We have demonstrated an effective algorithm for removing ringing artifacts from coded images and videos. The proposed filter is based on the conditional replacement filter (CRF) and takes into account the direction of the patterns it is filtering to reduce the risk of blurring. Objective results show a bit-rate reduction between 4% and 8% on video sequences.

## References

- [1] J.-M. Valin, *A Deringing Filter for Daala... And Beyond*, 2016. [https://people.xiph.org/~jm/daala/deringing\\_demo/](https://people.xiph.org/~jm/daala/deringing_demo/)
- [2] Median Filter, Wikipedia. [https://en.wikipedia.org/wiki/Median\\_filter](https://en.wikipedia.org/wiki/Median_filter)
- [3] C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images", *Proceedings of IEEE International Conference on Computer Vision*, 1998.
- [4] Daala website, Xiph.Org Foundation. <http://xiph.org/daala/>
- [5] Daala Git repository. <http://git.xiph.org/?p=daala.git;a=summary>
- [6] Are We Compressed Yet? <https://arewecompressedyet.com/>
- [7] T. Daede, J. Moffitt, *Video Codec Testing and Quality Measurement*, IETF Internet draft, 2015. <https://tools.ietf.org/html/draft-daede-netvc-testing>